

HI-TECH C Compiler for PIC10/12/16 MCUs Version 9.82 Release Notes

Produced on: June 14, 2011

Australian Design Centre
45 Colebard Street West
Acacia Ridge QLD 4110
Australia

web: http://www.htsoft.com

THIS FILE CONTAINS IMPORTANT INFORMATION RELATING TO THIS COMPILER. PLEASE READ IT BEFORE RUNNING THIS SOFTWARE.

Chapter 1

Introduction

1.1 Description

This 9.82 version of the compiler is a minor update, which has support for the latest devices and fixes bugs reported in previous versions.

An earlier 9.70 compiler release replaced both the previous HI-TECH C Compiler for PIC10/12/16 MCUs (formally the PRO compiler) and the HI-TECH PICC STD Compiler for the Microchip PIC micro. The PICC STD compiler will no longer be developed but will continue as an operating mode of this, and subsequent, compiler releases.

1.2 Further Help

The compiler manual covers all aspects of the compiler's operation, as well as other useful information. Check the well populated index for your search term.

Common problems are explained in the FAQ list. You can also ask questions of other users of this product in the forums.

1.3 Updates and Feedback

Microchip welcomes bug reports, suggestions or comments regarding this compiler version. Please direct any bug reports or feature requests via email to (support).

1.4 Previous Versions

The previous version of HI-TECH C Compiler for PIC10/12/16 MCUs was 9.81, released in January 2011. The previous HI-TECH PICC STD Compiler for the Microchip PICmicro was 9.60PL3, released in January 2009.

Chapter 2

Stop Press

The following are changes made that have not been released in the user's guide.

2.1 New Compiler Messages

The following PIC10/12/16 messages are new and do not appear in the manual.

(1397) unable to implement non-stack call to "*"; possible hardware stack overflow (Code Generator)

The compiler must encode a C function call without using a CALL assembly instruction and the hardware stack (and instead use a lookup table), but is unable to. A call might be required if the function is called indirectly via a pointer, but if the hardware stack is already full, an additional call will cause a stack overflow.

(1401) eeprom qualified variables cannot be accessed from both interrupt and mainline code (Code Generator)

All eeprom variables are accessed via routines which are not reentrant. Code might fail if an attempt is made to access eeprom variables from interrupt and main-line code. Avoid accessing eeprom variables in interrupt functions.

(1402) a pointer to eeprom cannot also point to other data types (Code Generator)

A pointer cannot have targets in both the eeprom space and ordinary data space.

(1404) unsupported: * (Parser)

The `unsupported __attribute__` has been used to indicate that some code feature is not supported. The message printed will indicate the feature that is not supported.

(1406) auto eeprom variables are not supported

Variables qualified as `eeprom` cannot be `auto`. You can define `static` local objects qualified as `eeprom`, if required.

(1407) bit eeprom variables are not supported

Variables qualified as `eeprom` cannot have type `bit`.

Chapter 3

New Features

The following are new features the compiler now supports. The version number in brackets indicates the first compiler version to support the feature.

3.1 General

HTML diagnostic output (9.82) The compiler can now produce a set of HTML files that may be of assistance in understanding the compilation process and build results for the project. Use the `--HTML` option (at all stages of compilation), or select the `Create html files` checkbox in the Linker tab of the Build Options dialog in MPLAB IDE. Look for the main page called `index.html` in the `html` subdirectory written to the project directory. (If output has been directed to a different directory, the `html` directory will appear in the specified directory.) The main page is a graphical representation of the compilation process. Each file icon is clickable and will open with the contents of that file (even intermediate files, and binary files open in a human-readable form), and each application icon can also be clicked to show a page containing information about that application's invocation and results.

New devices supported (9.82) The 12LF1840T48A device is now supported by the compiler, as well as both the F and LF variants of 16F1507, 10F320 and 10F322.

Updated MPLAB IDE plugin (9.82) A new plugin is installed with this compiler. The main change relates to memory that is reserved when a debugger is selected. This information now comes from the IDE rather than the compiler's INI file. The new plugin identifies itself as version 1.39. You can verify the version of a plugin from the Driver tab in the MPLAB IDE Build Options dialog.

New devices supported (9.81) The following devices are now supported by the compiler: 12F752, 12HV752, 16F1782, 16F1783, 16LF1782, 16LF1783, 16LF1904, 16LF1906, 16LF1907, MCV18E, RF675AF, RF675AH,

RF675AK, RF675F, RF675H and RF675K.

Supported device list (9.81) The file `chipinfo.html` contains a list of all device supported by the compiler. It is located in the `docs` directory of the compiler. This list can also be obtained in the usual way via the `--CHIPINFO` option, or from the MPLAB IDE Driver tab in the Build Options dialog.

3.2 Code Generator/Parser

eeprom qualifier (9.82) The `eeprom` qualifier has been reintroduced allowing named variables to be placed into the device's EEPROM space, if applicable. The `__EEPROM_DATA` macro is also still available for preloading this memory space. Note the following caveats that deal with `eeprom` qualified variables.

- Variables that are not initialized are not cleared at startup.
- Variables that are initialized will be preloaded with their initial value. This is a preload of the data rather than an assignment done at runtime, so after a software or hardware reset the content of the variables will not change and they may not hold their specified initial value.
- Access of these variables is extremely slow which typically prohibits the use of these variables in ordinary computations.
- These variables are accessed via special routines that are not reentrant. You cannot access these variables from both main-line and interrupt code. It is recommended that you do not access any `eeprom` qualified variable in an interrupt function or code the interrupt function calls.
- Not all expressions are possible using these variables.
- The bit variables cannot be qualified as `eeprom`.
- Any pointer that has a target qualified eeprom cannot also have a target in any other memory space.

Bank selection after in-line assembly code (9.80) When the compiler sees inline assembly, it will now assume that this code has affected the state of the currently selected bank.

3.3 Assembler

Better optimization for devices with one page (9.82) Optimization of assembly code that calls routines is improved for devices that only have one page of program memory. Less page selection instructions will be used for any routine called on these devices.

Chapter 4

Changes

The following are features that are now handled differently by the compiler. These changes may require modification to your source code if porting code to this compiler version. The version number in brackets indicates the first compiler version to implement the change.

4.1 General

Code size (9.82) As a result of bug fixes made in this release, the size of existing code may increase. For projects that are close to using all the device memory, this increase may result in can't find space errors from the linker or code generator.

New header file generation (9.81) Header files in the 9.81 version are now generated from a central database rather than being hand crafted. This may mean that the names of some SFRs or bits within SFRs in the new header files may be different to those in the header files that were previously used by the compiler.
The header files that were previously shipped with the compiler are no longer maintained, but are supplied with this version of the compiler and can be used by defining the macro `_LEGACY_HEADERS`. Ideally, this macro should be defined using the compiler's `-D` option, or in the Define macros field in the Compiler tab of the MPLAB IDE Build Options dialog. If you use a `#define` to define this, make sure you place this above the line which includes `<htc.h>`. The legacy include files are located in the legacy directory, inside the include directory of the compiler.

Manual (9.81) The user's manual has been updated and is now revision B of Document number DS51865.

4.2 Code Generator

Constant propagation in custom program memory psects (9.81) If `const`-qualified objects were placed into a user-defined psect (using the `#pragma psect` directive) the compiler may have applied its usual constant propagation optimizations on values read. This is usually of no concern, but where the memory accessed can be changed in other ways, this can lead to code failure. The compiler now treats any object in user-defined psects as if it was also marked as being `volatile`.

Context switch code (9.81) The context switch code now uses a SWAPF (rather than MOVF) instruction to preserve and restore the STATUS register in the context switch. This will mean that the contents of the STATUS register will be available to use inside the ISR, if required.

4.3 Header Files

Inclusion of assembly header files (9.82) If a device-specific assembly header is manually included, the header files will automatically include the generic header file (either `aspic.h` or `caspic.h`) rather than just produce a warning.

TRISGPIO register (9.82) SFR variables with the names TRIS and TRISGPIO are now defined for Baseline PIC devices that specify a GPIO register. These variables are identical and access the same TRIS register.

Chapter 5

Limitations

The following are limitations in the compiler's operation. These may be general coding restrictions, or deviations from information contained in the user's manual.

5.1 General

Delays and the watchdog timer Note that if the `_delay` feature is used to insert a simple delay sequence or loop, the CLRWDT instruction may be used in this sequence. If the watchdog timer is not being used, this is of no concern. If the watchdog timer is enabled, then be aware that delays may reset its timer.

Memory Report for Absolute Variables If the user defines absolute variables (variables placed at an absolute address via the `@` construct), and any of these variables overlap other variables in memory, the compiler will also count overlapping variables when it comes to reporting on memory usage. That is, if two, byte-sized variables are located at the same address, the report will indicate 2 bytes being used, rather than 1.

Sample Code The same code provided with the compiler is for reference only. It may not be appropriate for all devices and situations. Use this code as the basis for your own programs, however ensure that it is reviewed before compiling.

5.2 MPLAB IDE

Out of bound variables in watch view If variables are defined for Enhanced Mid-range devices that are large and placed in the linear memory space, MPLAB IDE shows these objects are being "out of bound". This is display issue and is not indicative of code failure.

Wrong watch values The IDE expects debug information to be available in COFF. This file format does not allow the compiler to specify the necessary information for the IDE to be able to correctly interpret some pointer variables. The compiler allocates pointer sizes based on their usage, which can make this issue seem "hit and miss". Any attempt to examine pointers in the IDE's Watch view may result in incorrect values being shown, or an "invalid" message being displayed in the view. These issues will correct themselves when ELF is employed to convey debug information in future IDE/compiler versions.

5.3 C Code

EEPROM variable limitations There are some limitations to using variables qualified as eeprom. See the section on page 7for more information.

Looping around allocGlobals error This rare error may be produced by the compiler under very specific conditions. There is presently no workaround for this issue, but changing the offending line of code can be tried. Instances that have triggered this issue range from functions returning NULL pointers, to code which requires allocation of a temporary variable.

Constant index into int array If a constant is used as the index into an array of `const int` type, the element read may be incorrect. This issue does not affect arrays of different types, or when the index is a variable.

Psect pragma If the `#pragma psect` directive is used to rename an existing `text` psect (when, for example, a function is to be positioned at a specific location in memory) the renamed psect may still be merged with other psects by the assembler optimizer. The merged psect may assume a different name to the renamed psect and the linker may produce an error if the renamed psect was referenced in an explicit linker option. You can check in the assembly list file to see if merging took place and the name of the merged psect. The merging can be prevented by disabling the assembler optimizations. Alternatively, the function can be made absolute to position it at a specific memory location.

Stack overflow When the managed stack is used (the `stackcall` suboption to the `--RUNTIME` option is enabled) in some situations the stack may overflow leading to code failure. With this option enabled, if a function call would normally overflow the stack, the compiler will automatically swap to using a lookup table method of calling the function to avoid the overflow. However, if these functions are indirect function calls (made via a pointer) the compiler will actually encode them using a regular call instruction and when these calls return, the stack will overflow. The managed stack works as expected for all direct function calls, and for all indirect calls that do not exceed the stack depth.

Main function size If the function `main` produces assembly code that is 0x1FF words long, this cannot be placed in the program memory and a "can't find space" error message is produced. Increasing or decreasing the size of the function will allow it to be positioned correctly and the error will not be displayed. This only affects Baseline PIC devices.

Ragged arrays inside structure When an array of pointers to const objects is made part of a structure, the size of the pointers used to access each element of the array may have the wrong size and result in erroneous access to the elements.

Functions called from in-line assembly The code generator will not be able to identify a C function called only from in-line assembly code*** if the definition for that C function is placed before the assembly call instruction in the source file. Placing the function definition after the call is acceptable. If the function cannot be identified, no code will be generated for the function and the linker will issue undefined symbol errors***.

Can't Generate Code messages When compiling for baseline devices, some complex expressions may cause compile-time errors `(712) Can't generate code for this expression`. The expressions should be simplified to work around this. This may require the use of additional variables to store intermediate results. This is most likely with long integer or floating-point arithmetic and particularly those devices with less than 4 bytes of common memory available.

Indirect function calls The parameters to functions called indirectly via a pointer may not be passed correctly if the following conditions are met.

- A function is called indirectly from both main-line and interrupt code
- This function has parameters
- The function pointer used to call this function is also assigned the address of other functions at some point in the program; and
- This function pointer is initialized when it is defined.

All other instances of indirectly calling a function, even from interrupt and main-line code, will work as expected.

option and tris For baseline devices, the OPTION and TRIS registers must be read/written as a byte. Reading or writing individual bits is not supported.

PIC17 support PIC 17 devices (for example, 17C756) is not supported by this compiler.

fast32 floats The option to select the fast 32-bit float or double library for PIC17 devices that was included in the PICC STD compiler is no longer available.

5.4 Assembler

Procedural abstraction An instance has been seen, but not resolved, which appears to be related to the procedural abstraction optimization of the assembler. If suspect behaviour is encountered, compiling for speed, instead of space, will disable this optimization and allow you to confirm if this is the underlying cause. This problem was reported for an Enhanced Mid-range device.

5.5 Libraries

Flash Read for 12F617 The FLASH_READ function cannot be used with this device as the routine uses different registers to those available on this device.

Peripheral routines The flash read/write routines/macro included with this compiler are functional only for devices that can write one word at a time (e.g. 16F877).

5.6 Header Files

Configuration bits for 16(L)F1936/37 The DEBUG configuration bit is missing from the device header files.

SFR definitions For the 16(L)F720/721, the following SFR definitions may be incorrect or missing; SSPADD, ADD<7:0>, MSK<7:0>. For the 16(L)F1825/1829, the following SFR definitions may be incorrect or missing; FVRCON, SRRC2E, SRSC2E, APFCON0, WPUB, ADD<7:0>, MSK<7:0>.

Chapter 6

Bug Fixes

The following are corrections that have been made to the compiler. These may fix bugs in the generated code or alter the operation of the compiler to that which was intended or specified by the user's manual. The version number in brackets indicates the first compiler version to implement the fix.

6.1 General

Installation errors with OS X installer (9.82) Errors resulted if no installation directory was entered when prompted (i.e. the default directory was used). This has been corrected; however in most situations an explicit destination is specified, so this error was not likely occur.

Increased memory usage and other side effects when including assembly (9.82) If assembly source modules (not in-line assembly) were added to a project, and this assembly contained psects destined for program memory, the compiler would reserve an amount of memory for this psect, but link and allocate additional memory for the psect elsewhere. This meant that program memory would be consumed as twice the rate it should. In some cases, the additional memory being reserved might cause some other psects to be misplaced and fixup errors would result.

6.2 Code generator/Parser

Wrong initial values (9.82) When defining large initialized objects for enhanced mid-range devices, it was possible that the initial values would not be correctly assigned to the objects. This only affected enhanced Mid-range devices and objects that would be accessed using the linear addressing mode.

Removal of externally referenced function (9.82) If a function was not used in C code, but was accessed in assembly and this function returned a value larger than a byte, a compile "rewrite" error would result. This has been corrected and such code can be used to prevent unused functions from being removed by the compiler.

Errors with assignment of right shifts (9.82) Were a value was being right shifted by either a literal amount of 8, 16 or 24 bits, and the result was being directly assigned to an `unsigned long` variable, errors may have resulted if the original value being shifted was not of an `unsigned long` type.

Code failure after interrupt on enhanced mid-range devices (9.82) A small number of code sequences require the use of a temporary register (`btemp`). This register was not being saved prior to execution of interrupt code. Code that used this register in both main-line and interrupt code may have failed.

Bad pointer arithmetic (9.82) Where a byte sized offset was being added to a 2-byte address, the addition may have been incorrectly performed.

Bad code produced for conversion of boolean to integral type (9.82) When a boolean value was converted to an integral type and then used in a subsequent expression, the code may have overwritten a value held in WREG resulting in incorrect operation.

Detection of duplicate case labels (9.82) Any `switch` statement that had more than one `case` label with the same `case` value was not being flagged as erroneous. This condition now raises a compiler error.

Warning for misplaced absolute const objects (9.82) A warning is now generated if you attempt to define an absolute const object at an address that is outside the target device's program memory space.

Error for bit parameters (9.82) An error is now generated if you attempt to define a function that accepts parameters of type `bit`.

Wrong code executed in switch statement (9.82) If the compiler uses the "direct" strategy to encode a `switch` statement and the target device is an enhanced mid-range PIC with only one page of program memory, the code may have jumped to the wrong location or crashed entirely. Other `switch` strategies and devices were not affected by this issue.

Undefined symbol associated with function pointers (9.82) If a function pointer inside a structure is initialized, but never used, an undefined symbol error (for a symbol like `fp_funcName`) might have been produced. This error has been corrected.

Inaccuracies and warnings with in-line delays (9.82) For a small range of delay values (around 78 mSec) the delay time of the `_delay` in-line function would be extremely small. In other situations, an assembly warning indicating truncation of an operand value may have been emitted. Both the inaccuracies and warnings have been corrected.

Incorrect casting (9.81) Expressions that involved a subtraction and a cast from one type to another of different size may have taken place in any order leading to wrong values. The optimizations involved will no longer take place when there are different sized types in the expression.

Conditional operator operands with side effects (9.81) The code generator was not properly identifying that operands to the conditional operator had side effects (e.g. they used the increment or decrement operators). In such cases, the code may fail as the operand expressions (with their side effects) were being executed regardless of the value of the control expression. Operands with side effects are now correctly identified, which will prevent any optimizations associated with the operator.

Parser crash with in-line assembly (9.81) If the C18-style in-line assembly was used with this compiler, e.g. `_asm()` (NB the leading underscore), the parser would crash. This has been resolved. Code should continue to use the usual `asm("instruction");` syntax for in-line assembly.

Integer to pointer conversion (9.81) In some instances, code that converts integers to pointers may cause the code generator to hang.

Spurious unused variable warnings (9.81) Symbols that are defined in library functions and which are not used will no longer trigger unused variable warnings.

Optimization with volatiles (9.81) The code generator was performing an optimization on a `volatile` symbol that resulted in that symbol being read prior to executing the expression in a preceding `if()` statement. When the `if()` expression also involved the same symbol, side effects could lead to code failure. Being `volatile`, this optimization should not have been performed and is now prevented.

Ungraceful exit with minimal memory settings (9.81) If the amount of RAM for an enhanced mid-range device was adjusted so that it was extremely small (i.e. most memory was reserved) the code generator might crash when calculating ranges for the linear memory ranges. This has been corrected and the code generator will handle this situation correctly.

Stack overflow (9.80) The code generator didn't take into account some levels of hardware stack required below a function when calculating the 'opt stack' control value for each function. As a result the assembler would perform procedural abstraction where there could potentially be no levels of stack available at runtime and a stack overflow result.

Bad code involving shifts (9.80) Code that involves shifts of byte-sized objects may have destroyed the contents of WREG and resulted in incorrect results. This has been corrected.

Use of temporary variable and interrupts (9.80) A temporary variable (`btemp`) was being used by complex statements in main-line code. This variables was also used in interrupt routines which could result in its contents being destroyed (and code failure) if an interrupt occurred. Such variables are no longer used by main-line code.

Structures containing pointer arrays (9.80) The storage size for pointers in an array that is itself a structure member may not have be correct. This would then result in errors when the pointers were accessed. Pointer size is now correctly determined.

6.3 Assembler

Wrong line numbers for error messages (9.82) The assembler was issuing error and warning messages that did not correspond to the assembly source file being processed. This only occurred if the assembly file was preprocessed (this will be the case for MPLAB IDE projects, unless this option has been turned off). In addition, the first message produced was accurate, but any subsequent messages stated the line number and filename of the preprocessed assembly file, not the source file. Although the line numbers of the problem in the preprocessed file were accurate, this was not particularly useful. All warning and error messages now correspond to the assembly source file.

Bad optimizations around branches (9.82) In rare circumstances, code may have been removed by the assembler optimizer. This would occur where instructions were moved "up" (lower address) in the assembly code, and inadvertently moved past a GOTO or CALL instruction. The optimization will no longer move code past any instruction that jumps or calls.

Confirmation of instruction availability (9.82) Checks against the target device were not being made for some enhanced mid-range instructions. Hand-written assembly code that used these instructions may have successfully compiled for target devices whose instruction set did not contain the instructions. An error will now be reported for any code that uses these instructions when they are not available.

Crash with long symbols (9.81) When the identifier length was increased well above the default value and long symbol names were being used, these symbols may have caused a buffer overrun in the assembler which could have crashed this application. This has been corrected and long symbols should work as expected.

Bad optimizations involving PAGESEL (9.80) The assembler optimizer was not correctly identifying the PAGESEL pseudo-op as an instruction that could change the PCLATH register. This may have lead to bad optimizations.

Assembler hangs (9.80) The assembly may have become stuck in an endless loop when processing psects which are absolute (use the `abs` flag). This was caused by two opposing optimizations which undid the work of the other. These optimizations will no longer operate on absolute psects.

Optimizer destroys WREG contents (9.80) In some instances, the assembler optimizer may not see that a CALLW instruction (and the code it calls) destroys the contents of WREG, and results in a previous load of WREG being destroyed. This has now been corrected.

6.4 Driver

Bad memory ranges (9.80) Where absolute ROM memory ranges (i.e. ranges which did not add to, or subtract from, the default ranges) were correctly specified by the programmer (including page boundaries), these ranges may have been concatenated in such a way that page boundaries relevant to the target device where removed. This may have resulted in code that crossed these boundaries and either compile- or runtime-errors. The page boundaries are now correctly preserved. NB The programmer must specify any memory page boundaries that are relevant to the target device when using this option.

Successful build for code with errors (9.80) If the parser's exit status reported an error and the `--pass1` option was is in effect, the residual p1 file was not being deleted. Build systems (e.g. make, MPLAB IDE) which rely on date-time stamps were seeing the existence of this file to indicate that the preliminary build step was successful. Now, the file is deleted and IDEs forced to rebuild the intermediate p1 file.

6.5 Linker and Utilities

Can't find space messages (9.82) There was inconsistencies in one of the metrics used by the linker to determine the order in which psects were allocated memory. This has been corrected, may result in a different allocation order of psects, and may stop some "can't find space messages" that linker issued.

Cromwell crash (9.82) Cromwell may have crashed in some situations when the compiled code contained absolute bits with internal linkage.

Cromwell error with 10LF devices (9.82) Cromwell would produce an error when compiling for any 10LF device. The 10F devices were not affected by this issue.

Cromwell crash (9.80) In some instances where structures were used before their definition, the resulting diagnostic files may have caused the Cromwell application to crash. This situation has been resolved.

6.6 Libraries and Header Files

Failure to include assembly header files (9.82) If a C module included <htc.h> and also included <caspic.h>, the latter would not be included. The code to prevent multiple exclusions was not correct. This has been rectified and both headers can be included into the same module.

Missing address masking in bit definitions (9.82) The `BANKMASK()` macro is now used with definitions of bit operands within SFRs defined in the assembly version of the device header files. The omission of this mask meant that fixup errors may have occurred for hand-written assembly code that used the bit definitions from either the `aspic.h` or `caspic.h` header files.

Missing OPTION2 register (9.82) The definition for the `OPTION2` register was missing from the device header files for the 16HV540 device. Accessing this register in C code will produce assembly code that uses the special `TRIS` instruction to access the register contents.

Multiple inclusion of header files (9.81) A bug in the scripts that generate the `aspic.h` header file allowed the device-specific assembly header files to be included more than once. The script has been corrected so that this will no longer occur.

Missing ltoa library function (9.80) The `ltoa` library function was not compiled into the supplied libraries, resulting in errors when trying to use this routine. The source code is now built into the libraries.